CMPT 365 Multimedia Systems

Final Review - 1

Spring 2017

Outline

- Entropy
- Lossless Compression
  - Shannon-Fano Coding
  - Huffman Coding
  - LZW Coding
  - Arithmetic Coding
- Lossy Compression
  - Quantization
  - Transform Coding - DCT

Why is Compression possible ?

- Information Redundancy

- Question: How is "information" measured ?

Self-Information

Information is related to probability
Information is a measure of uncertainty (or "surprise")

- Intuition 1:
  - I've heard this story many times vs This is the first time I hear about this story
  - Information of an event is a function of its probability:

    $i(A) = f(P(A))$. Can we find the expression of $f()$?
- Intuition 2:
  - Rare events have high information content
    - Water found on Mars!!!
  - Common events have low information content
    - It's raining in Vancouver.

  → Information should be a decreasing function of the probability:
  Still numerous choices of $f()$.
- Intuition 3:
  - Information of two independent events = sum of individual information:

    If $P(AB)=P(A)P(B)$ ➔ $i(AB) = i(A) + i(B)$.

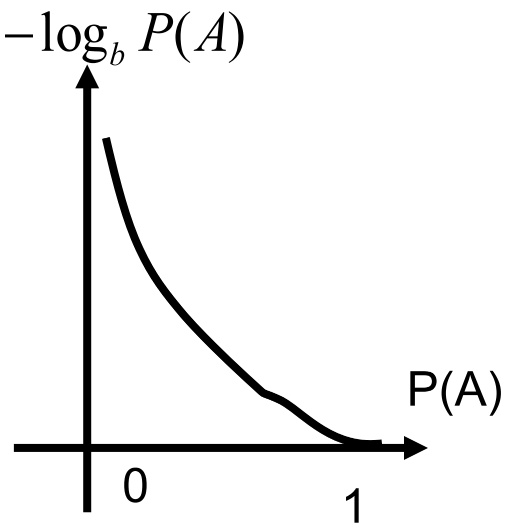
  → Only the logarithmic function satisfies these conditions.

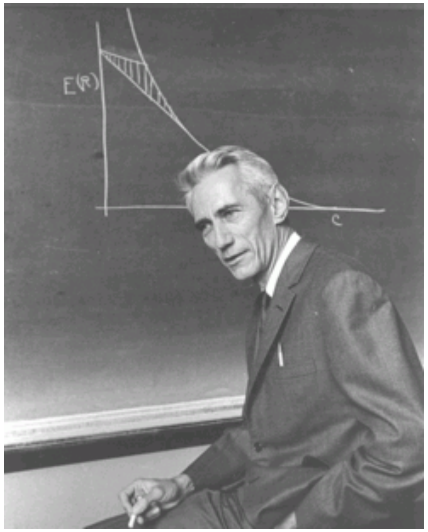
Self-information

- Shannon's Definition [1948]:
  - Self-information of an event:

$$i(A) = \log_b \frac{1}{P(A)} = -\log_b P(A)$$

If b = 2, unit of information is bits





Entropy

- Suppose:
  - a data source generates output sequence from a set $\{A_1, A_2, \ldots, A_N\}$
  - $P(A_i)$: Probability of A_i
- First-Order Entropy (or simply Entropy):
  - the average self-information of the data set

$$H = \sum_i -P(A_i)\log_2 P(A_i)$$

- The first-order entropy represents the minimal number of bits needed to losslessly represent one output of the source.

Example 1

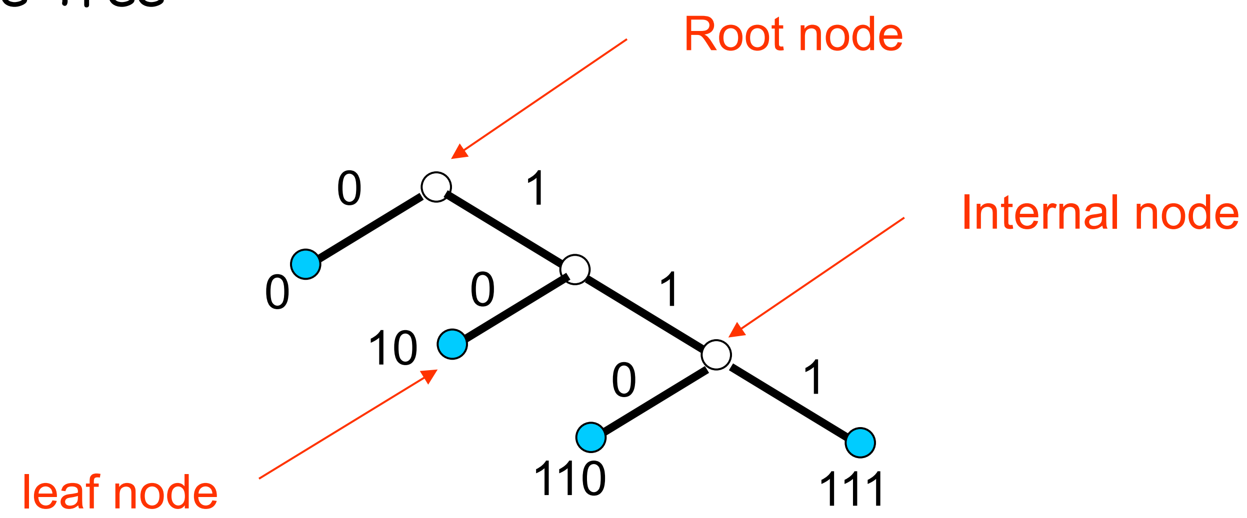
- X is sampled from {a, b, c, d}
- Prob: {1/2, 1/4, 1/8, 1/8}
- Find entropy.

Outline

- Why compression ?
- Entropy
- Variable Length Coding
  - Shannon-Fano Coding
  - Huffman Coding
  - LZW Coding
  - Arithmetic Coding

Entropy Coding: Prefix-free Code

- No codeword is a prefix of another one.
- Can be uniquely decoded.
- Also called prefix code
- Example: 0, 10, 110, 111
- Binary Code Tree



- Prefix-free code contains leaves only.
- How to state it mathematically?

Shannon-Fano Coding

- **Shannon-Fano Algorithm** - a top-down approach
  - Sort the symbols according to the frequency count of their occurrences.
  - Recursively divide the symbols into two parts, each with approximately the same number of counts, until all parts contain only one symbol.
- **Example: coding of "HELLO"**

Symbol	H	E	L	O
Count	1	1	2	1

Frequency count of the symbols in "HELLO"

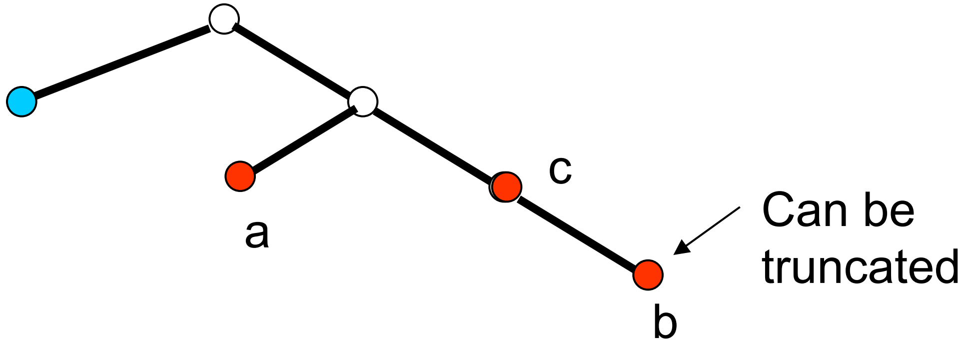
Outline

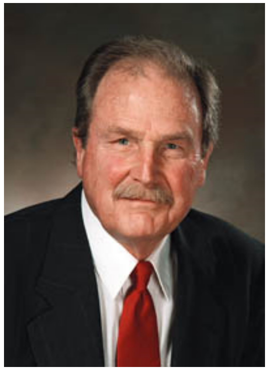
- Why compression ?
- Entropy
- Variable Length Coding
  - Shannon-Fano Coding
  - Huffman Coding
  - LZW Coding
  - Arithmetic Coding

Huffman Coding

- A procedure to construct optimal prefix-free code
- Result of David Huffman's term paper in 1952 when he was a PhD student at MIT

  Shannon → Fano → Huffman
- Observations:
  - Frequent symbols have short codes.
  - In an optimum prefix-free code, the two codewords that occur least frequently will have the same length.





Huffman Coding

- **Human Coding** - a bottom-up approach
  - Initialization: Put all symbols on a list sorted according to their frequency counts.
    - This might not be available !
  - Repeat until the list has only one symbol left:

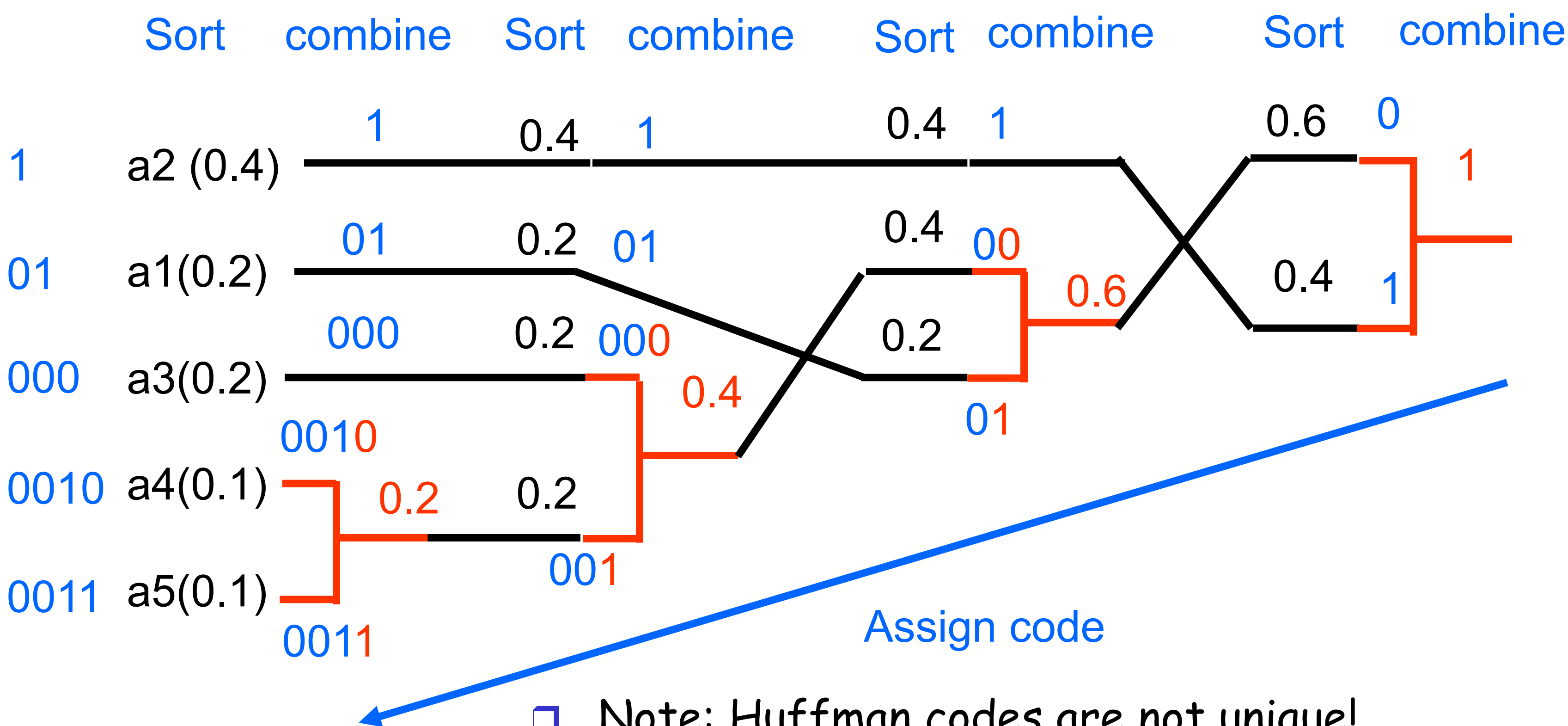
    (1) From the list pick two symbols with the lowest frequency counts. Form a Huffman subtree that has these two symbols as child nodes and create a parent node.

    (2) Assign the sum of the children's frequency counts to the parent and insert it into the list such that the order is maintained.

    (3) Delete the children from the list.
  - Assign a codeword for each leaf based on the path from the root.

More Example

- Source alphabet A = {a1, a2, a3, a4, a5}
- Probability distribution: {0.2, 0.4, 0.2, 0.1, 0.1}



- Note: Huffman codes are not unique!
  - Labels of two branches can be arbitrary.
  - Multiple sorting orders for tied probabilities

Properties of Huffman Coding

- **Unique Prefix Property**:
  - No Human code is a prefix of any other Human code - precludes any ambiguity in decoding.
- **Optimality**:
  - *minimum redundancy code* - proved *optimal* for a given data model (i.e., a given, accurate, probability distribution) under certain conditions.
  - The two least frequent symbols will have the same length for their Human codes, differing only at the last bit.
  - Symbols that occur more frequently will have shorter Huffman codes than symbols that occur less frequently.
- Average Huffman code length for an information source *S* is strictly less than *entropy*+ 1

$$\bar{l} < \eta + 1$$

Example

- Source alphabet A = {a, b, c, d, e}
- Probability distribution: {0.2, 0.4, 0.2, 0.1, 0.1}
- Code: {01, 1, 000, 0010, 0011}

- Entropy:

  $$\mathbf{H(S)} = -(0.2*\log_2(0.2)*2 + 0.4*\log_2(0.4)+0.1*\log_2(0.1)*2)$$
  $$= 2.122 \text{ bits / symbol}$$

- Average Huffman codeword length:

  $$\mathbf{L} = 0.2*2+0.4*1+0.2*3+0.1*4+0.1*4 = 2.2 \text{ bits / symbol}$$

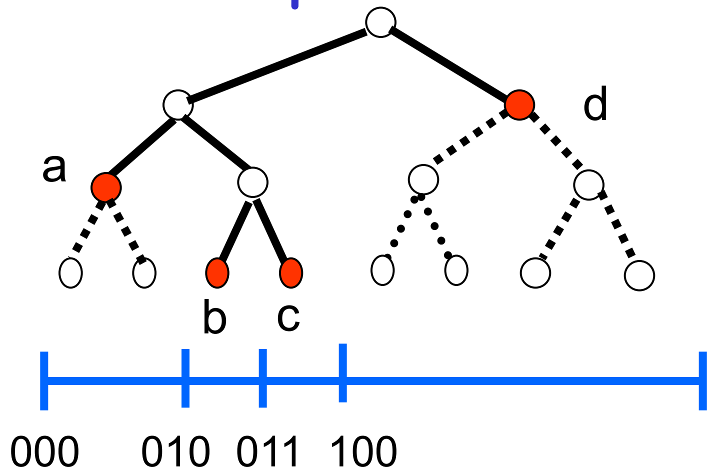
- In general: $H(S) \le L < H(S) + 1$

Huffman Decoding

- Direct Approach:
  - Read one bit, compare with all codewords…
  - Slow
- Binary tree approach:
  - Embed the Huffman table into a binary tree data structure
  - Read one bit:
    - if it's 0, go to left child.
    - If it's 1, go to right child.
    - Decode a symbol when a leaf is reached.
  - Still a bit-by-bit approach

Table Look-up Method

a: 00
b: 010
c: 011
d: 1




```
char HuffDec[8][2] = {
    {'a', 2},
    {'a', 2},
    {'b', 3},
    {'c', 3},
    {'d', 1},
    {'d', 1},
    {'d', 1},
    {'d', 1}
};
```

```
x = ReadBits(3);
k = 0;    //# of symbols decoded
While (not EOF) {
    symbol[k++] = HuffDec[x][0];
    length = HuffDec[x][1];
    x = x << length;
    newbits = ReadBits(length);
    x = x | newbits;
  x = x & 111B;
}
```

Extended Huffman Code

- Code multiple symbols jointly
  - Composite symbol: $(X_1, X_2, \ldots, X_k)$
  - Alphabet increased exponentioally: k^N
- Code symbols of different meanings jointly
  - JPEG: Run-level coding
  - H.264 CAVLC: context-adaptive variable length coding
    - # of non-zero coefficients and # of trailing ones
  - Studied later

Example

- $P(X_i = 0) = P(X_i = 1) = 1/2$
  - Entropy $H(X_i)$ = 1 bit / symbol
- Joint probability: $P(X_{i-1}, X_i)$
  - $P(0, 0) = 3/8$, $P(0, 1) = 1/8$
  - $P(1, 0) = 1/8$, $P(1, 1) = 3/8$
- Second order entropy:

Joint Prob $P(X_{i-1}, X_i)$

X_{i-1} \ X_i	0	1
0	3/8	1/8
1	1/8	3/8

$$H(X_{i-1}, X_i) = 1.8113 \text{ bits / 2 symbols, or } 0.9056 \text{ bits / symbol}$$

- Huffman code for Xi: 0, 1
- Average code length: 1 bit / symbol
- Huffman code for (X_{i-1}, X_i): 1, 00, 010, 011
- Average code length: 0.9375 bit /symbol

Consider 10 00 01 00 00 11 11 11 -- every two; non-overlapped

Outline

- Why compression ?
- Entropy
- Variable Length Coding
  - Shannon-Fano Coding
  - Huffman Coding
  - LZW Coding
  - Arithmetic Coding

LZW: Dictionary-based Coding

- LZW: Lempel-Ziv-Welch (LZ 1977, +W 1984)
  - Patent owned by Unisys http://www.unisys.com/about__unisys/lzw/
    - Expired on June 20, 2003 (Canada: July 7, 2004)
  - ARJ, PKZIP, WinZip, WinRar, Gif,
- Uses fixed-length codewords to represent variable-length strings of symbols/characters that commonly occur together
  - e.g., words in English text.
  - Encoder and decoder build up the same dictionary dynamically while receiving the data.
  - Places longer and longer repeated entries into a dictionary, and then emits the *code* for an element, rather than the string itself, if the element has already been placed in the dictionary.

LZW Algorithm

```
BEGIN
   s = next input character;
   while not EOF
   {
     c = next input character;

     if s + c exists in the dictionary
        s = s + c;
     else
        {
        output the code for s;
        add string s + c to the dictionary with a new code;
        s = c;
        }
   }
   output the code for s;
END
```

Example

- **LZW compression for string "ABABBABCABABBA"**
- Start with a very simple dictionary (also referred to as a "string table"), initially containing only 3 characters, with codes as follows:

code	string
1	A
2	B
3	C

- Input string is "ABABBABCABABBA"

```
BEGIN
   s = next input character;
   while not EOF
   {
     c = next input character;

     if s + c exists in the dictionary
        s = s + c;
     else
        {
        output the code for s;
        add string s + c to the
   dictionary with a new code;
        s = c;
        }
   }
   output the code for s;
END
```

s	c	output	code	string
			1	A
			2	B
			3	C
A	B	1	4	AB
B	A	2	5	BA
A	B			
AB	B	4	6	ABB
B	A			
BA	B	5	7	BAB
B	C	2	8	BC
C	A	3	9	CA
A	B			
AB	A	4	10	ABA
A	B			
AB	B			
ABB	A	6	11	ABBA
A	EOF	1		

- Input ABABBABCABABBA
- Output codes: 1 2 4 5 2 3 4 6 1. Instead of sending 14 characters, only 9 codes need to be sent (compression ratio = 14/9 = 1.56).

LZW Decompression (simple version)

```
BEGIN
   s = NIL;
   while not EOF
    {
     k = next input code;
     entry = dictionary entry for k;
     output entry;
     if (s != NIL)
        {add string s + entry[0] to dictionary with a new code;}

        s = entry;
    }
END
```

- **Example 7.3:** LZW decompression for string "ABABBABCABABBA".
- Input codes to the decoder are 1 2 4 5 2 3 4 6 1.
- The initial string table is identical to what is used by the encoder.

- The LZW decompression algorithm then works as follows:
- **Input: 1 2 4 5 2 3 4 6 1**

```
BEGIN
   s = NIL;
   while not EOF
    {
     k = next input code;
     entry = dictionary
    entry for k;
     output entry;
     if (s != NIL)
        add string s +
    entry[0] to dictionary
    with a new code;
        s = entry;
    }
END
```

S	K	Entry/output	Code	String
			1	A
			2	B
			3	C
NIL	1	A		
A	2	B	4	AB
B	4	AB	5	BA
AB	5	BA	6	ABB
BA	2	B	7	BAB
B	3	C	8	BC
C	4	AB	9	CA
AB	6	ABB	10	ABA
ABB	1	A	11	ABBA
A	EOF			

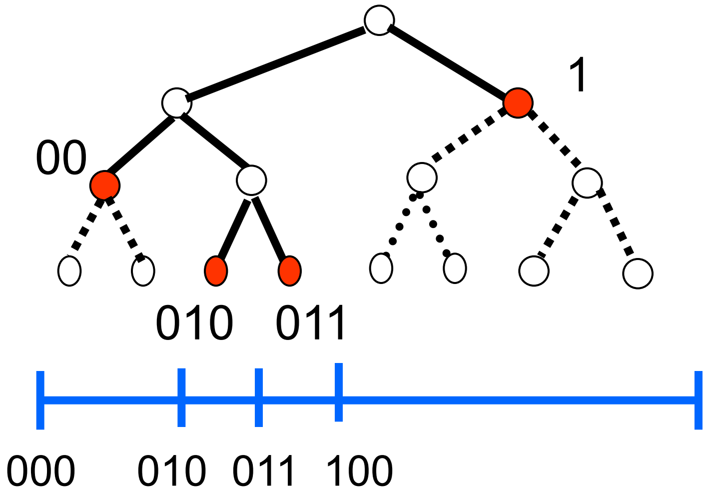
- Apparently, the output string is "ABABBABCABABBA", a truly lossless result!

Outline

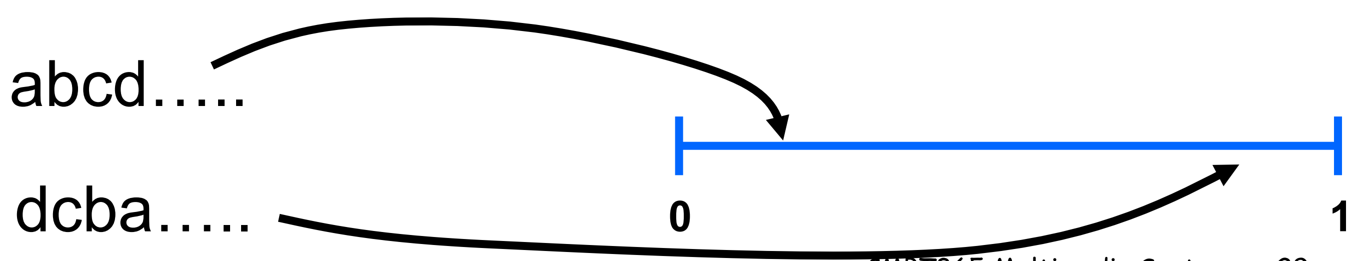
- Why compression ?
- Entropy
- Variable Length Coding
  - Shannon-Fano Coding
  - Huffman Coding
  - LZW Coding
  - Arithmetic Coding

Basic Idea

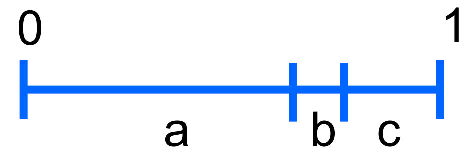
- Recall table look-up decoding of Huffman code
  - N: alphabet size
  - L: Max codeword length
  - Divide [0, 2^L] into N intervals
  - One interval for one symbol
  - Interval size is roughly proportional to symbol prob.



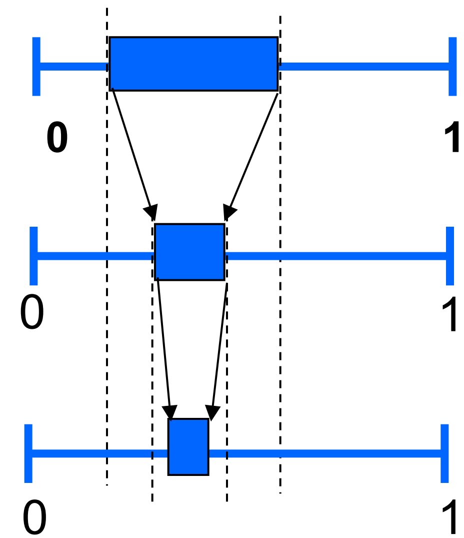
- Arithmetic coding applies this idea recursively
  - Normalizes the range [0, 2^L] to [0, 1].
  - Map a sequence to a unique tag in [0, 1).



Arithmetic Coding



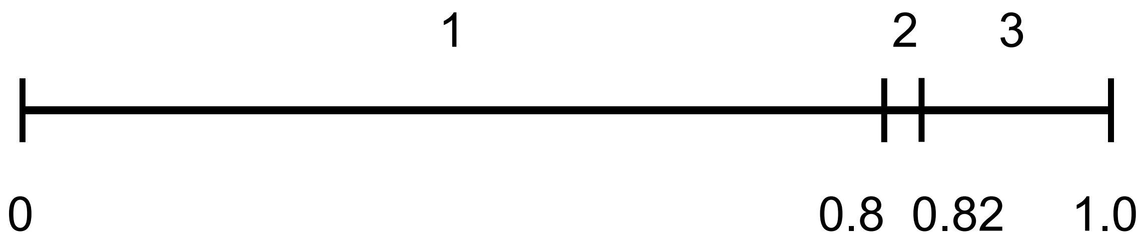
- Disjoint and complete partition of the range [0, 1)
  [0, 0.8), [0.8, 0.82), [0.82, 1)
- Each interval corresponds to one symbol
- Interval size is proportional to symbol probability



- The first symbol restricts the tag position to be in one of the intervals
- The reduced interval is partitioned recursively as more symbols are processed.
- Observation: once the tag falls into an interval, it never gets out of it

Example:

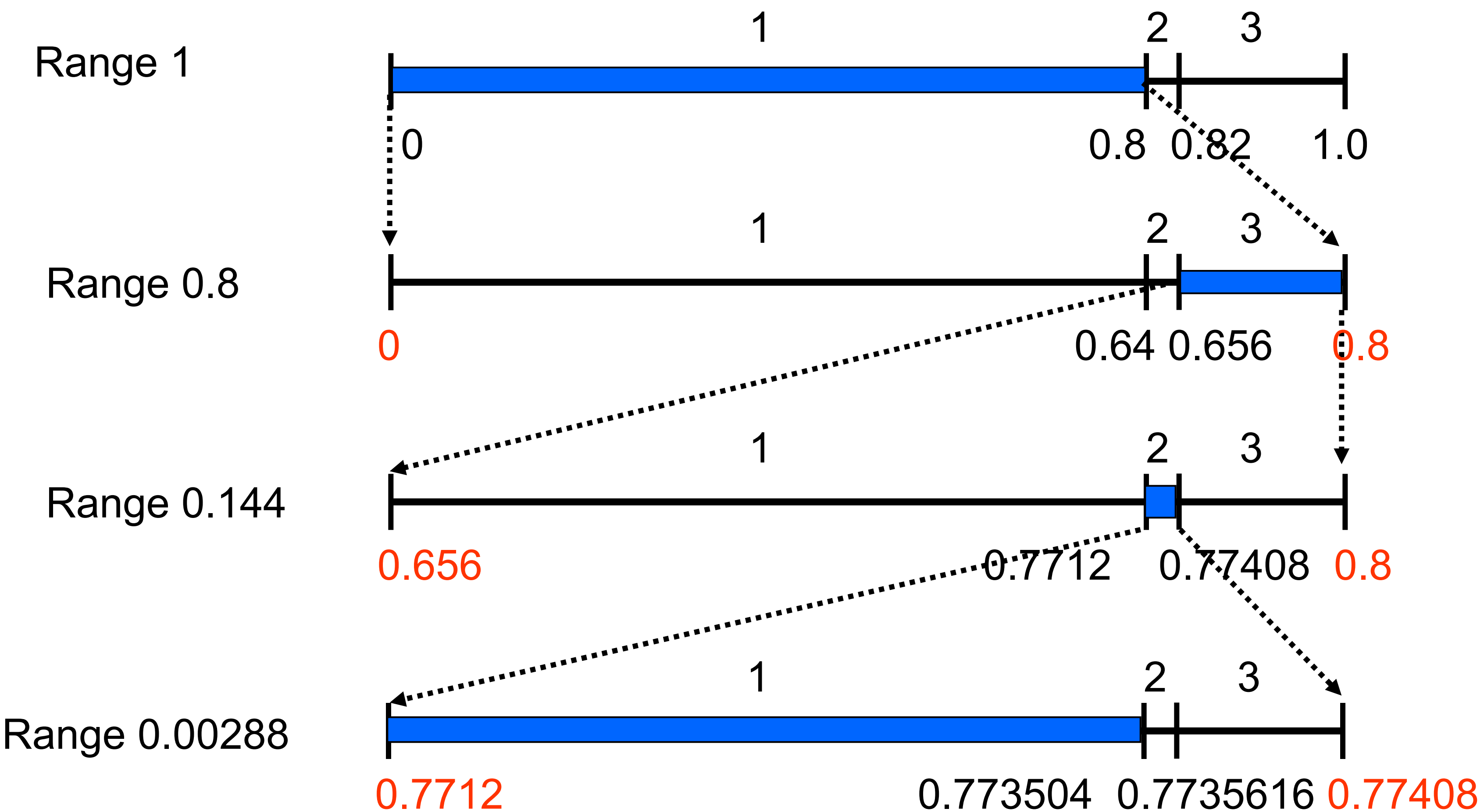
Symbol	Prob.
1	0.8
2	0.02
3	0.18



- Map to real line range [0, 1)
- Order does not matter
  - Decoder need to use the same order
- Disjoint but complete partition:
  - 1: [0, 0.8): 0, 0.799999…9
  - 2: [0.8, 0.82): 0.8, 0.819999…9
  - 3: [0.82, 1): 0.82, 0.999999…9
  - (Think about the impact to integer implementation)

Encoding

- Input sequence: "1321"



Final range: [0.7712, 0.773504): Encode 0.7712

Difficulties: 1. Shrinking of interval requires high precision for long sequence.
2. No output is generated until the entire sequence has been processed.

Encoder Pseudo Code

- Keep track of LOW, HIGH, RANGE
  - Any two are sufficient, e.g., LOW and RANGE.

```
BEGIN
low = 0.0;  high = 1.0;  range = 1.0;
while (symbol != terminator)
{
   get (symbol);
   low = low + range * Range_low(symbol);
   high = low + range *
   Range_high(symbol);
   range = high - low;
}
output a code so that low <= code < high;
END
```

Input	HIGH	LOW	RANGE
Initial	1.0	0.0	1.0
1	0.0+1.0*0.8=0.8	0.0+1.0*0 = 0.0	0.8
3	0.0 + 0.8*1=0.8	0.0 + 0.8*0.82=0.656	0.144
2	0.656+0.144*0.82=0.77408	0.656+0.144*0.8=0.7712	0.00288
1	0.7712+0.00288*0.8=0.773504	0.7712+0.00288*0=0.7712	0.002304

Generating Codeword for Encoder

```
BEGIN
   code = 0;
   k = 1;
   while (value(code) < low)
   {
        assign 1 to the kth binary fraction bit
        if (value(code) >= high)
               replace the kth bit by 0
        k = k + 1;
   }
END
```

- The final step in Arithmetic encoding calls for the generation of a number that falls within the range [*low*, *high*). The above algorithm will ensure that the shortest binary codeword is found.

Simplified Decoding

- Normalize RANGE to [0, 1) each time
- No need to recalculate the thresholds.

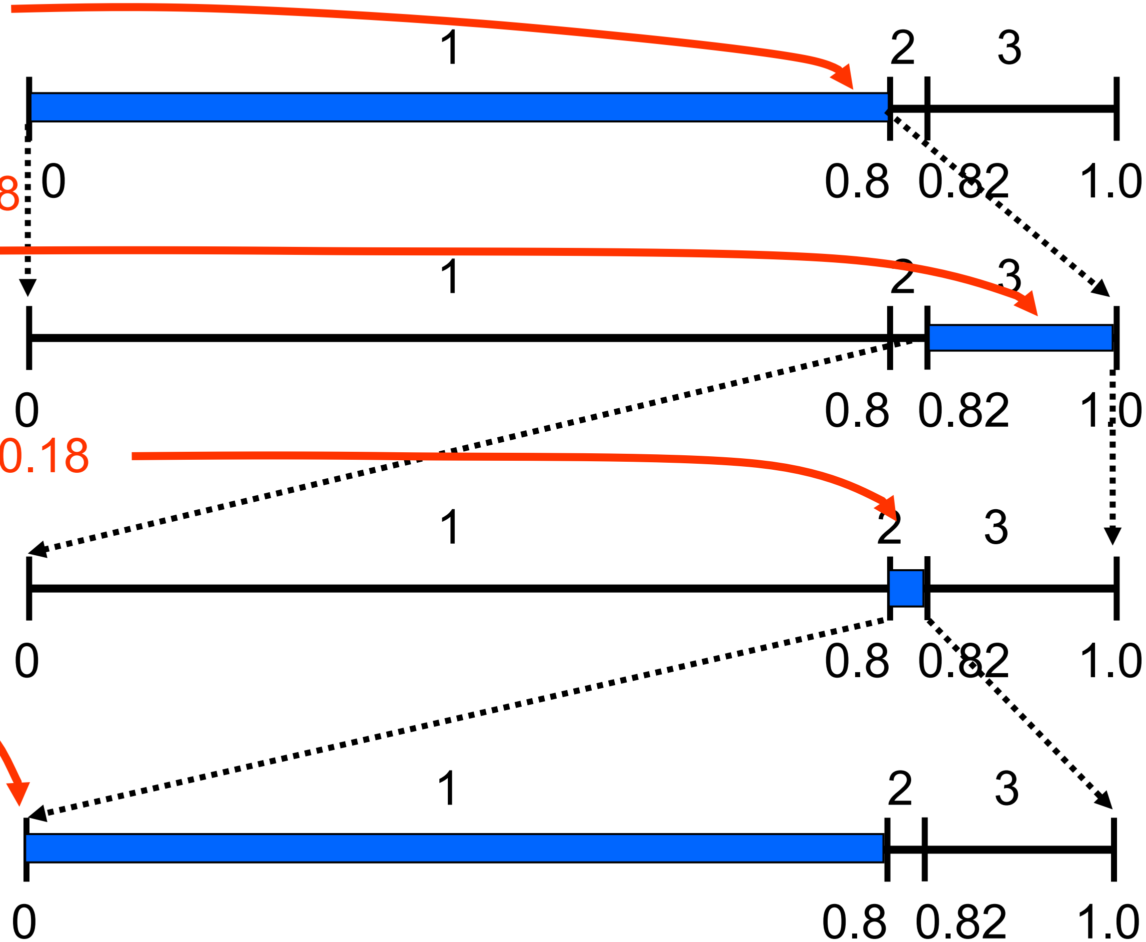
$$x \leftarrow \frac{x - low}{range}$$

Receive 0.7712
Decode 1

x =(0.7712-0) / 0.8
= 0.964
Decode 3

x =(0.964-0.82) / 0.18
= 0.8
Decode 2

x =(0.8-0.8) / 0.02
= 0
Decode 1



Decoder Pseudo Code

```
BEGIN
get binary code and convert to
decimal value = value(code);
DO
{
   find a symbol s so that
        Range_low(s) <= value < Range_high(s);
   output s;
   low = Rang_low(s);
   high = Range_high(s);
   range = high - low;
   value = [value - low] / range;
}
UNTIL symbol s is a terminator
END
```

Lossless vs Lossy Compression

- If the compression and decompression processes induce no information loss, then the compression scheme is **lossless**; otherwise, it is **lossy**.
- Why is lossy compression possible ?

Original

Compression Ratio: 7.7

Compression Ratio: 12.3

Compression Ratio: 33.9

Outline

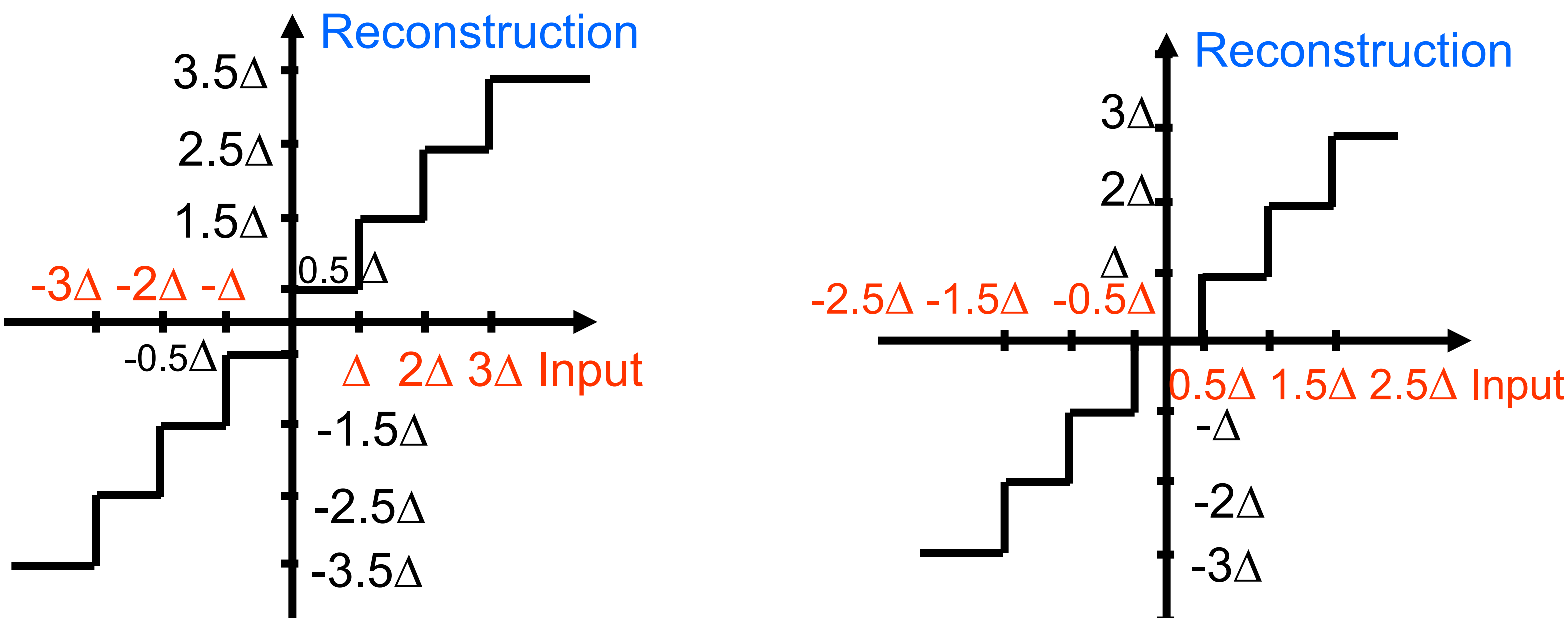
- Quantization
  - Uniform
  - Non-uniform
- Transform coding
  - DCT

Uniform Quantizer

- All bins have the same size except possibly for the two outer intervals:
  - bi and yi are spaced evenly
  - The spacing of bi and yi are both **Δ (step size)**

$$y_i = \frac{1}{2}\left(b_{i-1} + b_i\right)$$ for inner intervals.

Uniform Midrise Quantizer — Uniform Midtread Quantizer



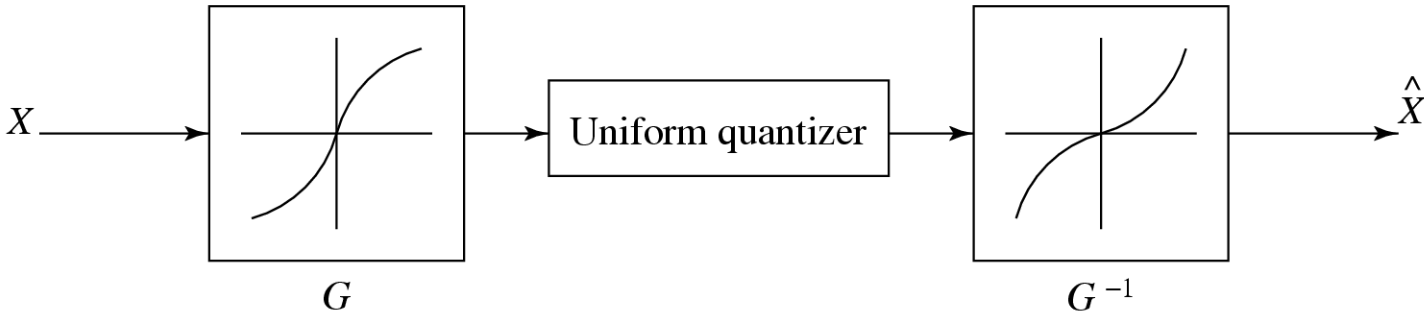
Measure of Distortion

- Quantization error: $e(x) = x - \hat{x}$
- Mean Squared Error (MSE) for Quantization
  - Average quantization error of all input values
  - Need to know the probability distribution of the input

- Number of bins: M
- Decision boundaries: b_i, i = 0, …, M
- Reconstruction Levels: y_i, i = 1, …, M
- Reconstruction: $\hat{x} = y_i \quad \text{iff } b_{i-1} < x \leq b_i$

- MSE: $$MSE_q = \int_{-\infty}^{\infty} (x - \hat{x})^2 f(x)dx = \sum_{i=1}^{M} \int_{b_{i-1}}^{b_i} (x - y_i)^2 f(x)dx$$
  - Same as the variance of e(x) if μ = E{e(x)} = 0 (zero mean).
  - Definition of Variance: $$\sigma_e^2 = \int_{-\infty}^{\infty} (e - \mu_e)^2 f(e)de$$

Non-uniform Quantization



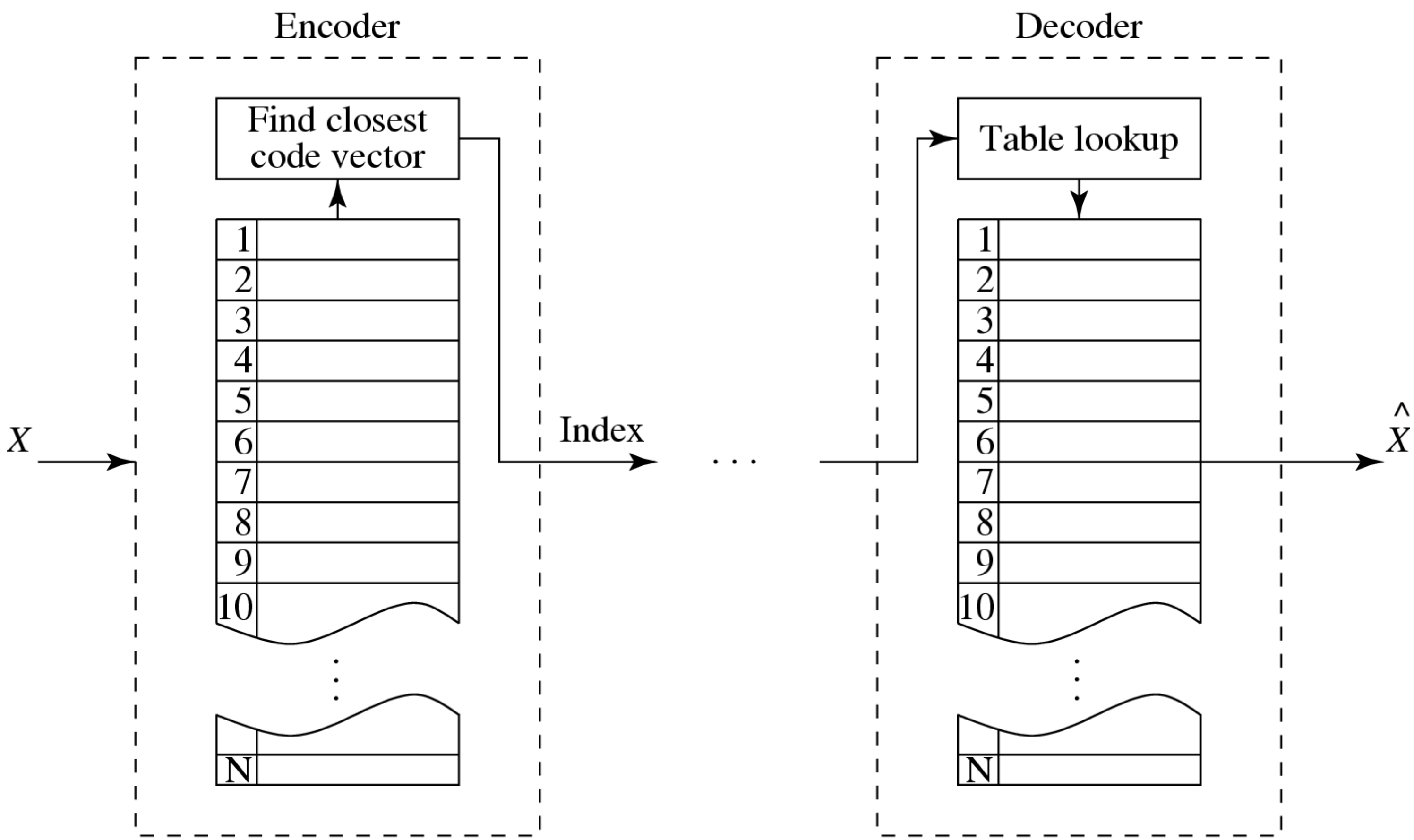
- *Companded quantization* is **nonlinear**.
- As shown above, a *compander* consists of a *compressor function* G, a uniform quantizer, and an *expander function* G^{-1}.
- The two commonly used companders are the *μ*-law and A-law companders.

Outline

- Quantization
  - Uniform
  - Non-uniform
  - Vector quantization
- Transform coding
  - DCT

Vector Quantization (VQ)

- According to Shannon's original work on information theory, any compression system performs better if it operates on vectors or groups of samples rather than individual symbols or samples.
- Form vectors of input samples by simply concatenating a number of consecutive samples into a single vector.
- Instead of single reconstruction values as in scalar quantization, in VQ code *vectors* with n components are used. A collection of these code vectors form the *codebook*.



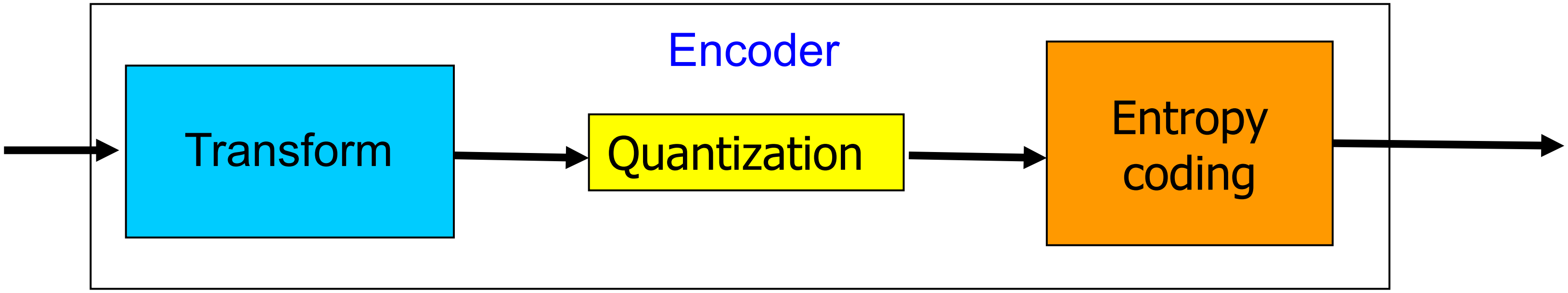
- **Fig. 8.5**: Basic vector quantization procedure.

Outline

- Quantization
  - Uniform quantization
  - Non-uniform quantization
- Transform coding
  - Discrete Cosine Transform (DCT)

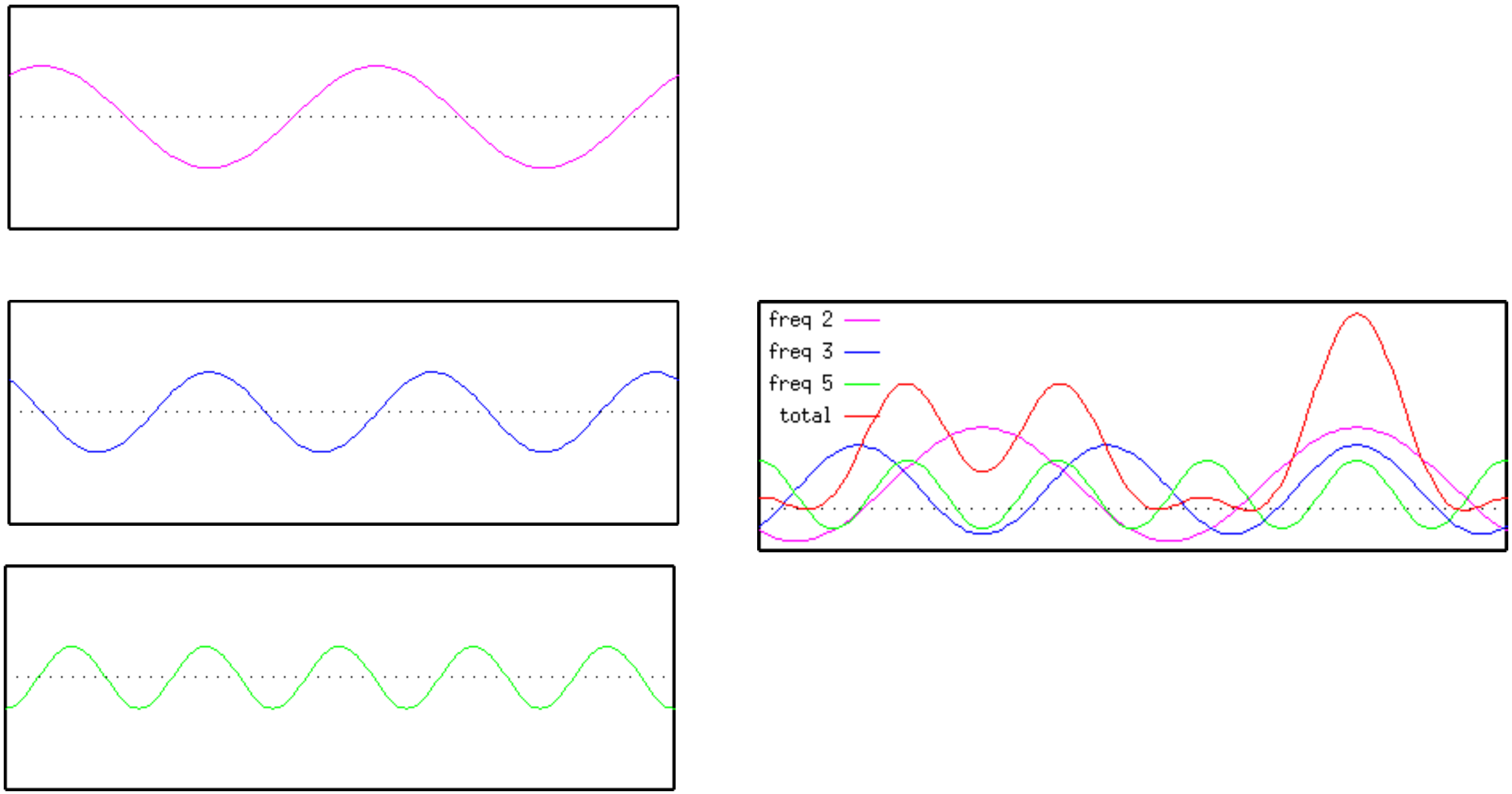
Why Transform Coding ?

- Transform
  - From one domain/space to another space
  - Time -> Frequency
  - Spatial/Pixel -> Frequency
- Purpose of transform
  - Remove correlation between input samples
  - Transform most energy of an input block into a few coefficients
  - Small coefficients can be discarded by quantization without too much impact to reconstruction quality



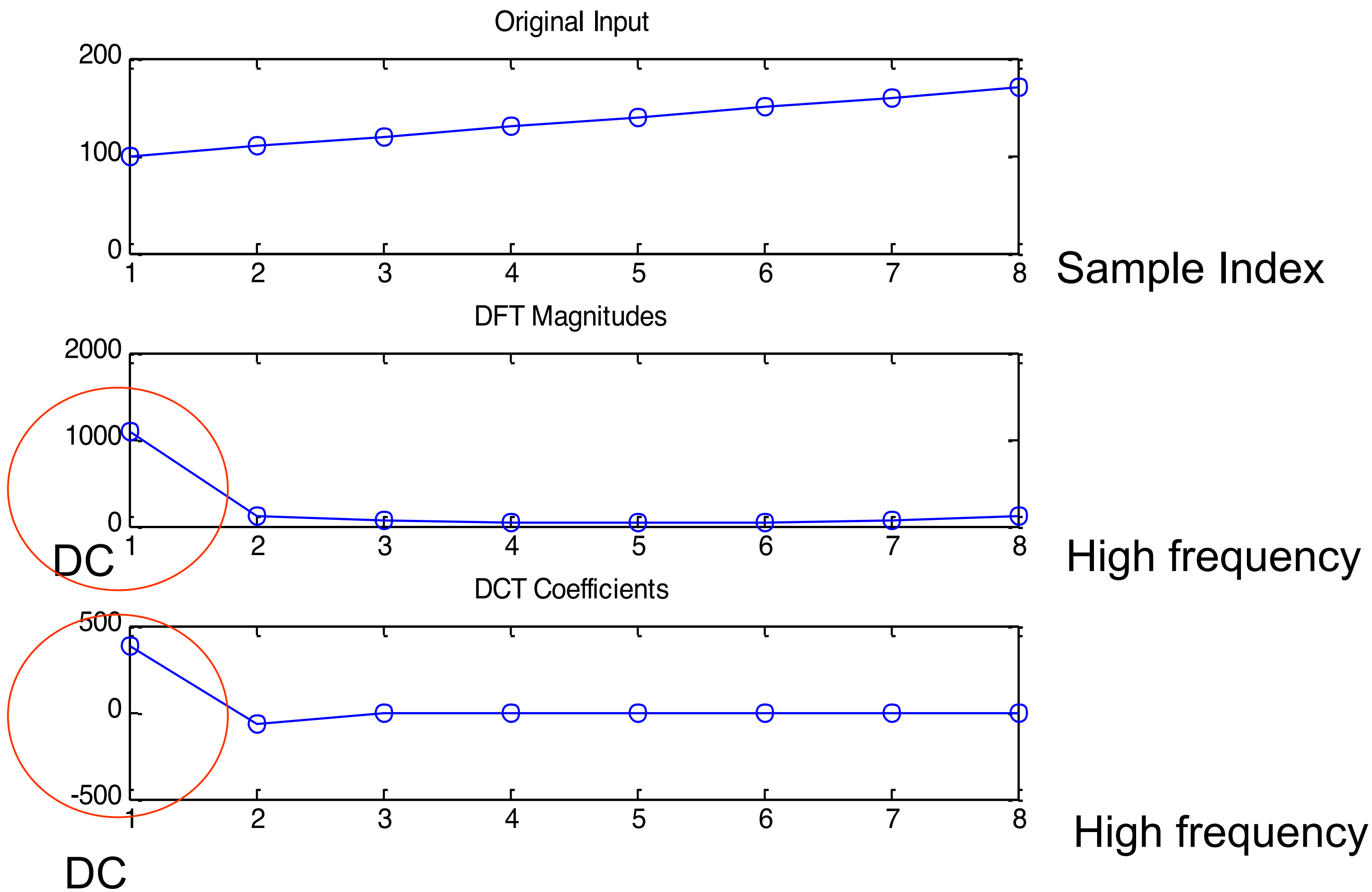
1-D Example

- Fourier Transform



1-D Example

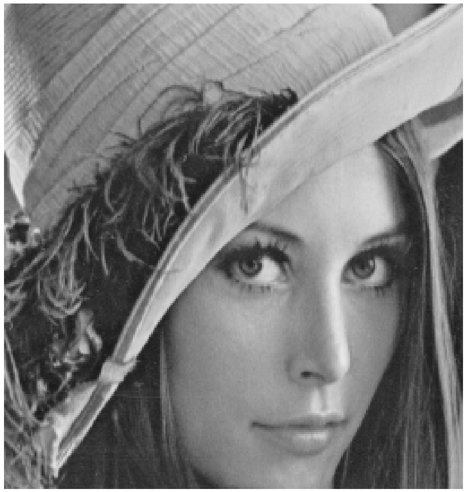
- Smooth signals have strong DC (direct current, or zero frequency) and low frequency components, and weak high frequency components

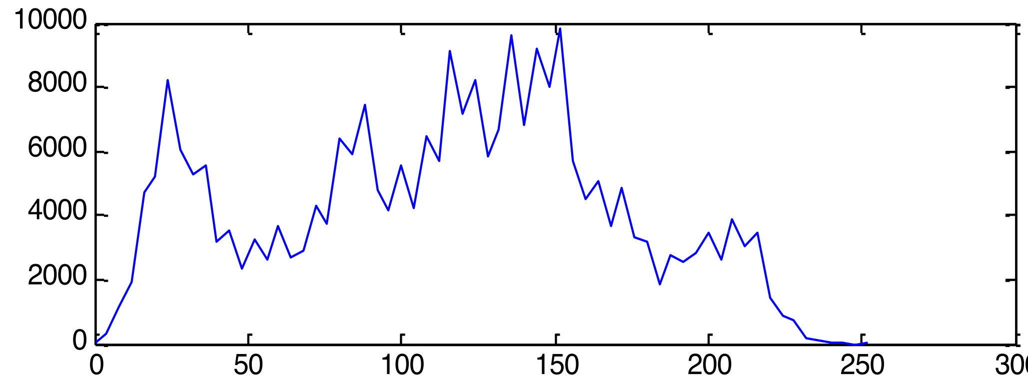


2-D Example

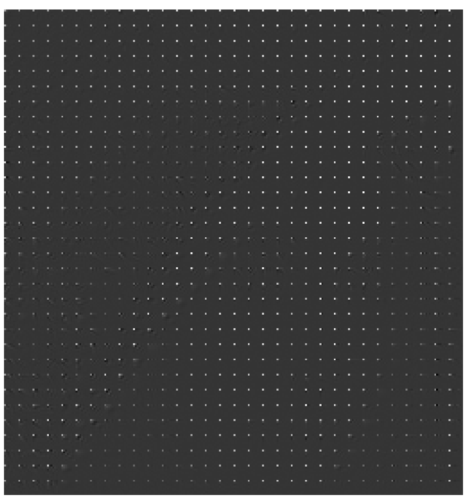
- Apply transform to each 8x8 block
- Histograms of source and DCT coefficients

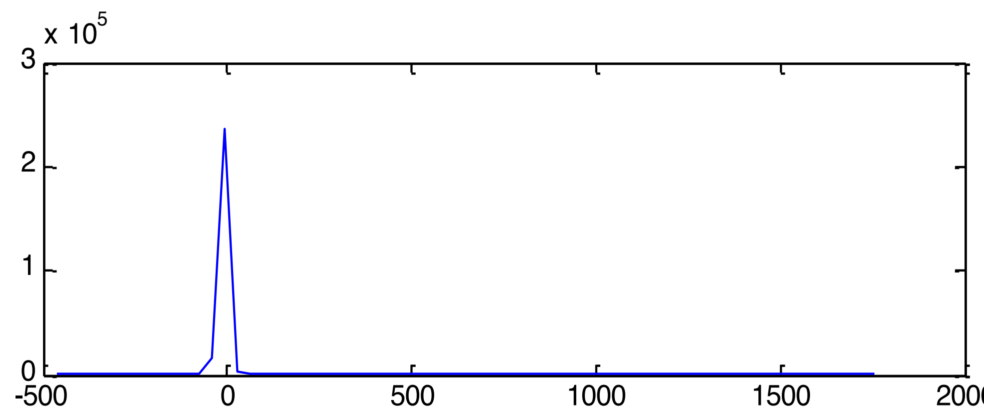
Original Image





2-D DCT Coefficients. Min= -465.37, max= 1789.00



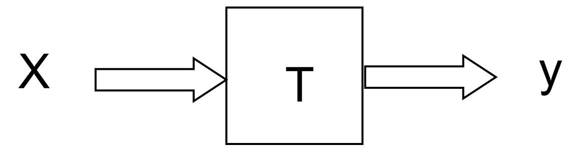


- Most transform coefficients are around 0.
- Desired for compression

Matrix Representation of Transform

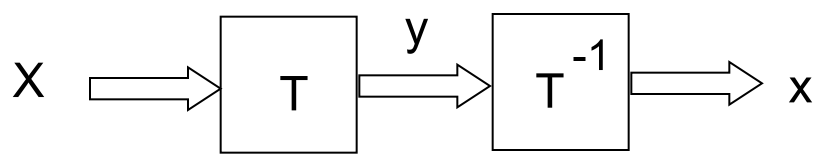
- Linear transform is an N x N matrix:

$$\mathbf{y}_{N\times 1} = \mathbf{T}_{N\times N}\mathbf{x}_{N\times 1}$$



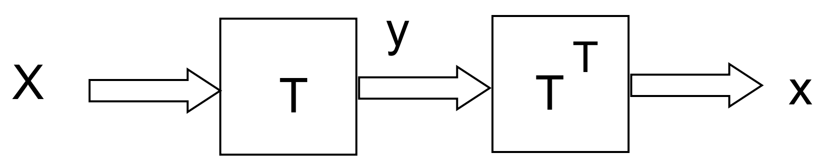
- Inverse Transform:

$$\mathbf{x} = \mathbf{T}^{-1}\mathbf{y}$$



- Unitary Transform (aka orthonormal):

$$\mathbf{T}^{-1} = \mathbf{T}^{T}$$



- For unitary transform: rows/cols have unit norm and are orthogonal to each others

$$\mathbf{T}\mathbf{T}^{T} = \mathbf{I} \;\Rightarrow\; \mathbf{t}_i\mathbf{t}_j^{T} = \delta_{ij} = \begin{cases} 1, & \mathrm{i} = \mathrm{j} \\ 0, & \mathrm{i} \neq \mathrm{j} \end{cases}$$

1D Discrete Cosine Transform (1D DCT):

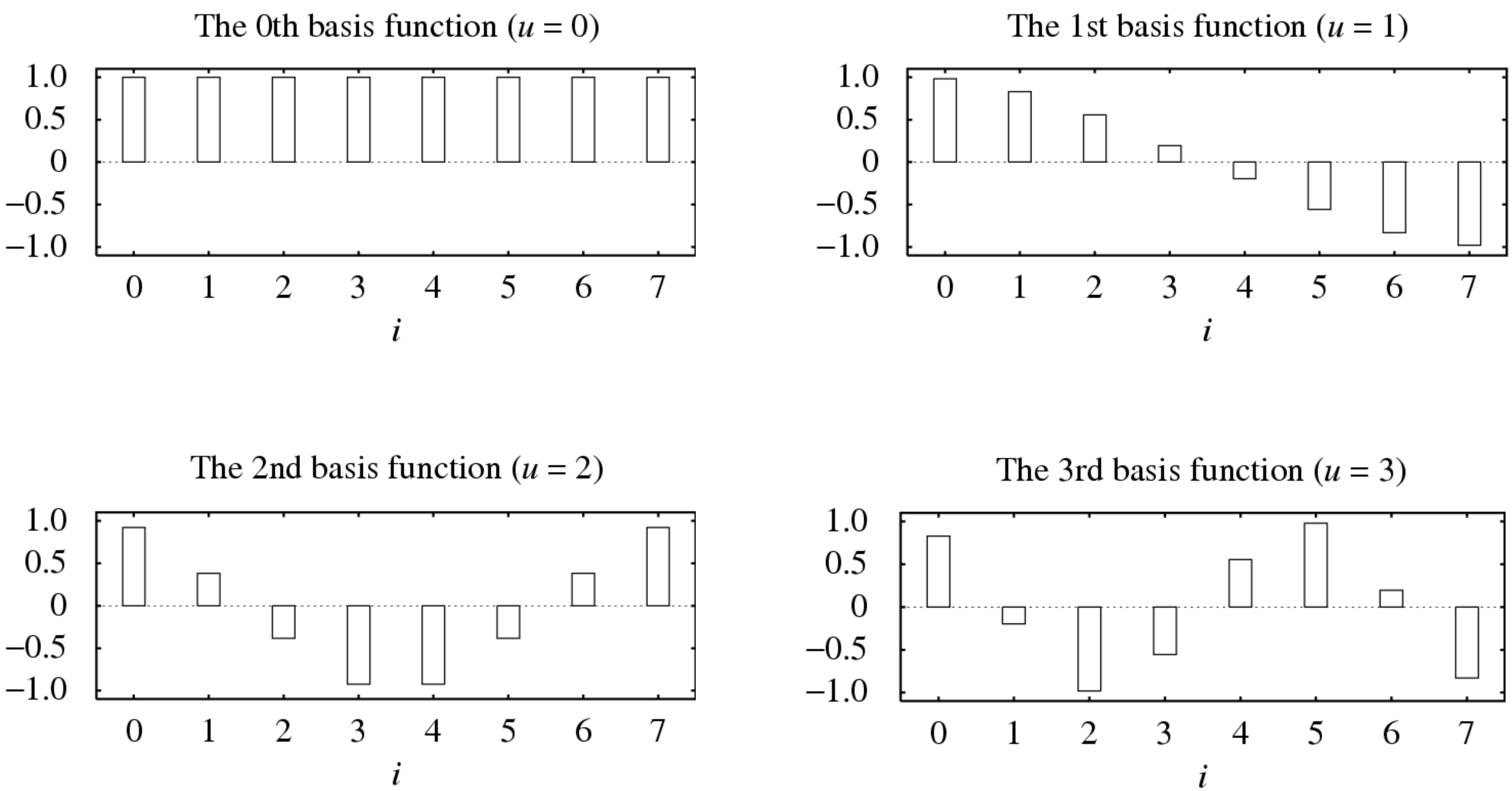
$$F(u) = \frac{C(u)}{2} \sum_{i=0}^{7} \cos \frac{(2i+1)u\pi}{16} f(i) \qquad (8.19)$$

- where i = 0, 1, . . . , 7, u = 0, 1, . . . , 7.

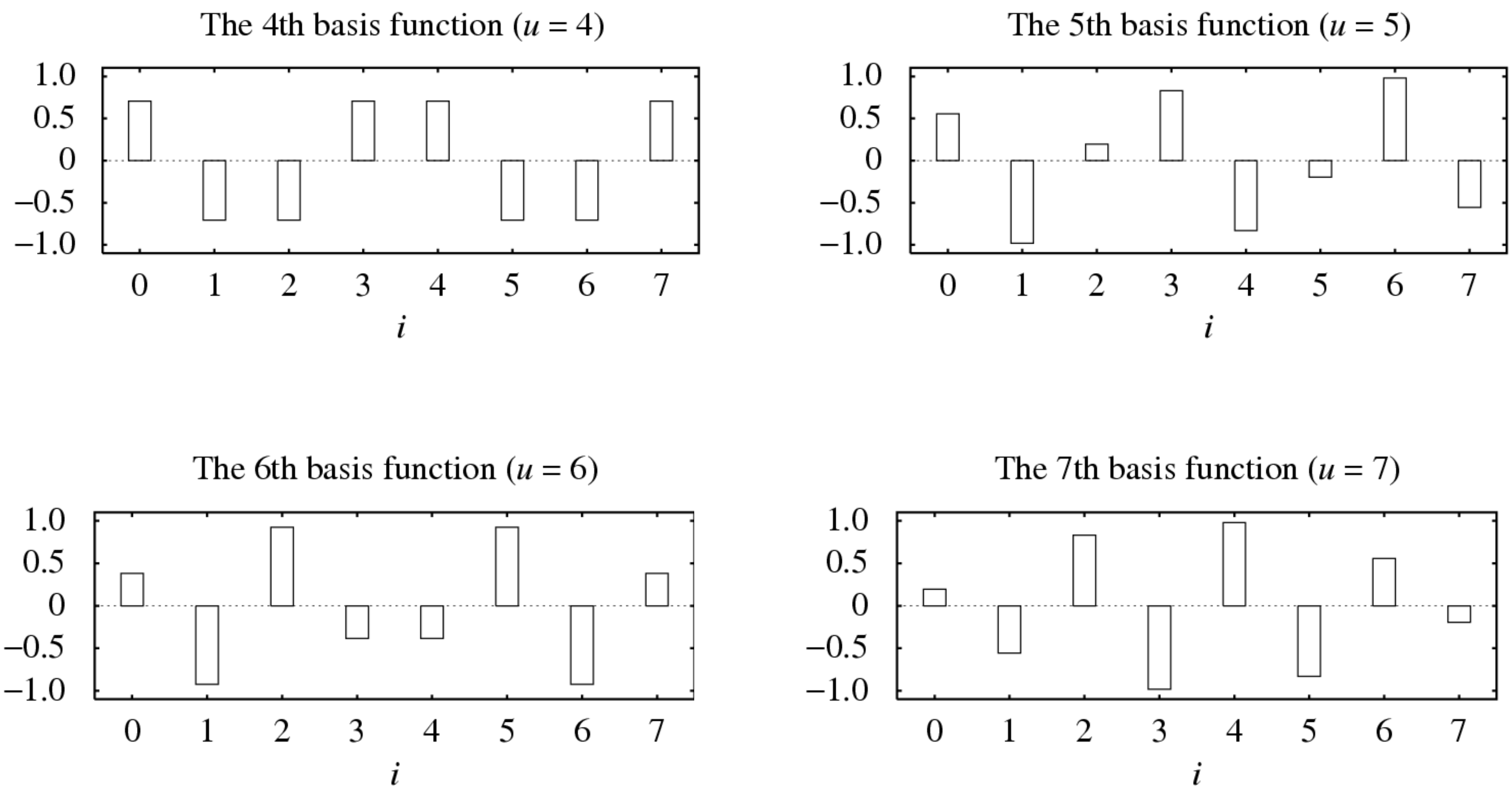
- **1D Inverse Discrete Cosine Transform (1D IDCT):**

$$\tilde{f}(i) = \sum_{u=0}^{7} \frac{C(u)}{2} \cos \frac{(2i+1)u\pi}{16} F(u) \qquad (8.20)$$

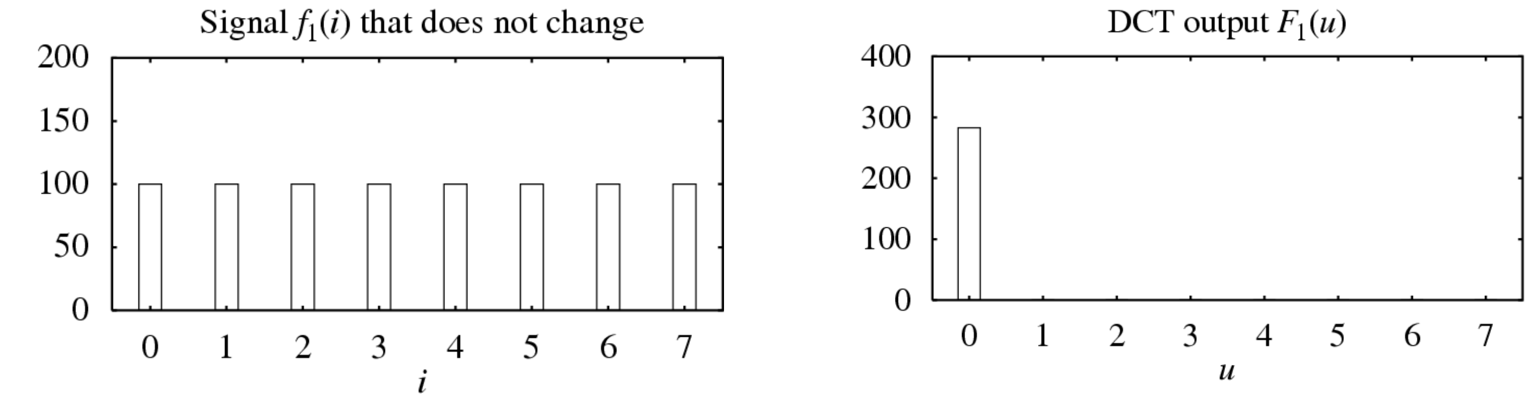
- where i = 0, 1, . . . , 7, u = 0, 1, . . . , 7.



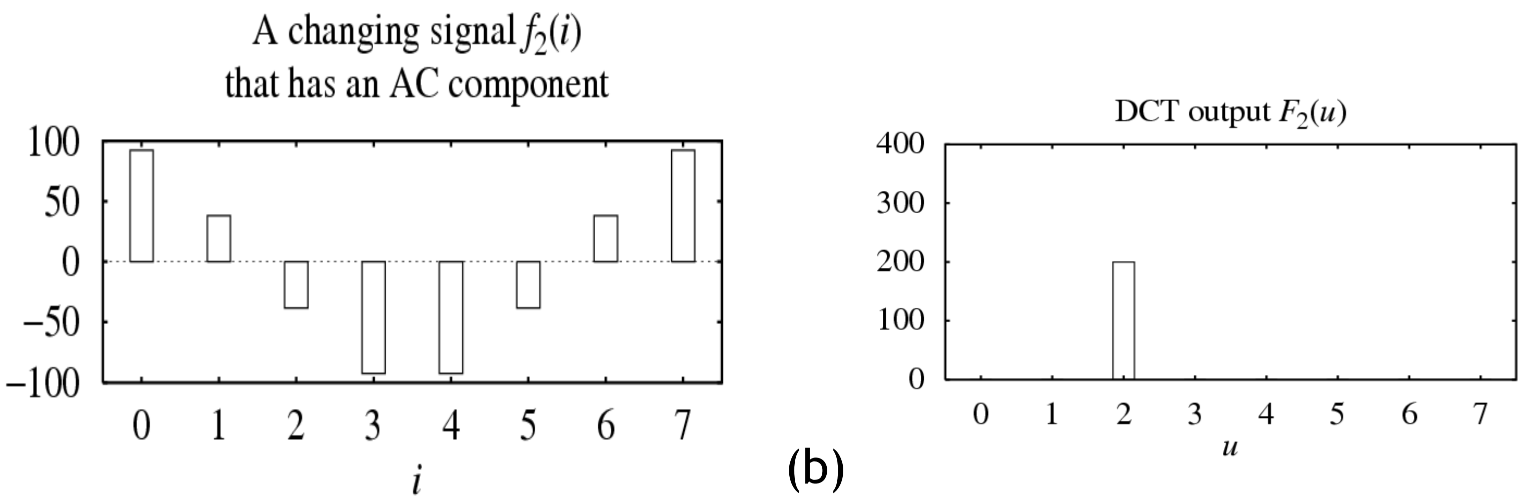
- **Fig. 8.6**: The 1D DCT basis functions.



- **Fig. 8.6 (Cont'd):** The 1D DCT basis functions.

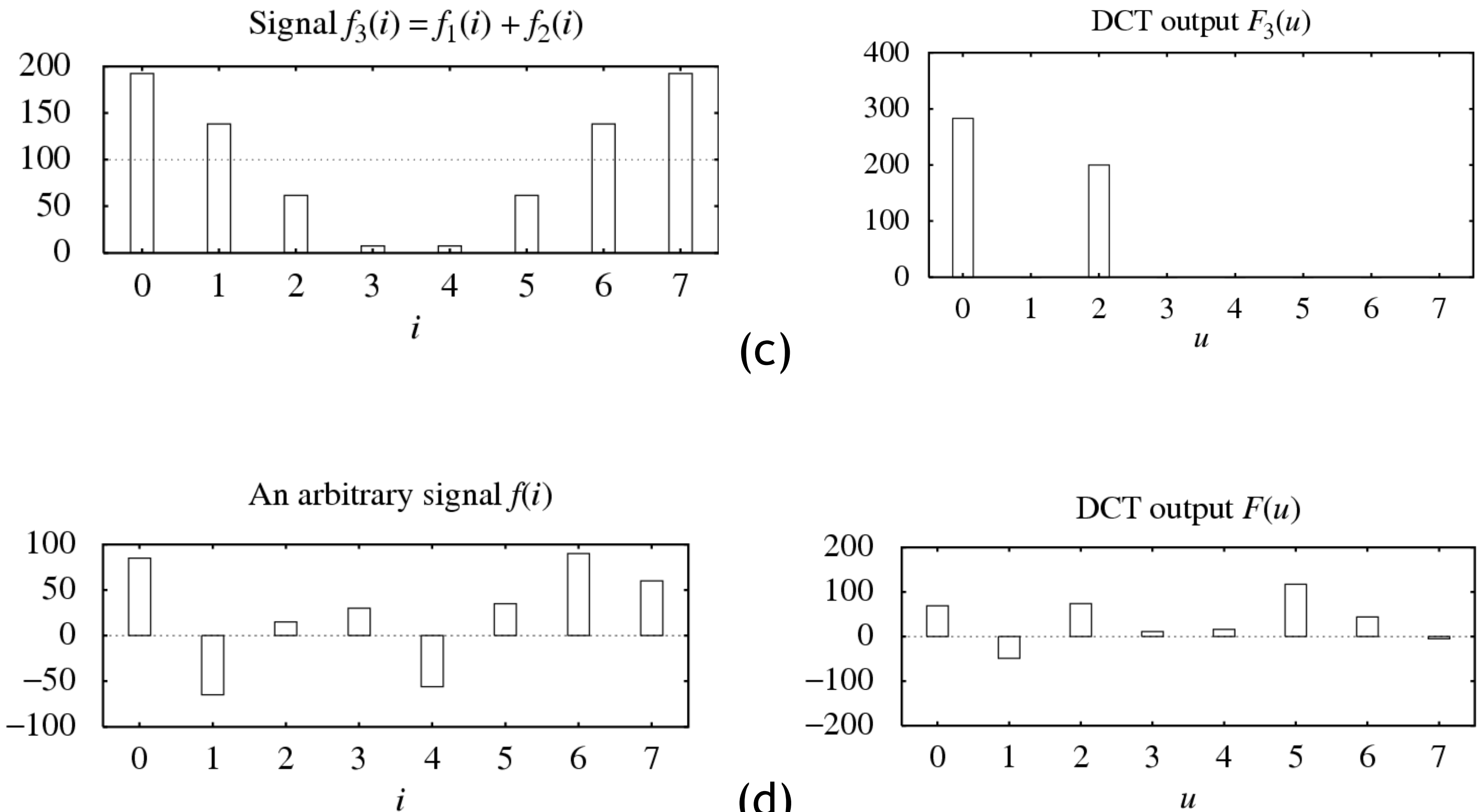


(a)



(b)

- **Fig. 8.7:** Examples of 1D Discrete Cosine Transform: **(a)** A DC signal $f_1(i)$, **(b)** An AC signal $f_2(i)$.



- **Fig. 8.7 (Cont'd):** Examples of 1D Discrete Cosine Transform: **(c)** $f_3(i) = f_1(i)+f_2(i)$, and **(d)** an arbitrary signal $f(i)$.

The Cosine Basis Functions

- Function $B_p(i)$ and $B_q(i)$ are *orthogonal*, if

$$\sum_i [B_p(i) \cdot B_q(i)] = 0 \qquad if \quad p \neq q$$

- (8.22)

- Function $B_p(i)$ and $B_q(i)$ are *orthonormal*, if they are orthogonal and

$$\sum_i [B_p(i) \cdot B_q(i)] = 1 \qquad if \quad p = q$$

- (8.23)

- It can be shown that:

$$\sum_{i=0}^{7} \left[\cos\frac{(2i+1) \cdot p\pi}{16} \cdot \cos\frac{(2i+1) \cdot q\pi}{16} \right] = 0 \qquad if \ p \neq q$$

- $$\sum_{i=0}^{7} \left[\frac{C(p)}{2} \cos\frac{(2i+1) \cdot p\pi}{16} \cdot \frac{C(q)}{2} \cos\frac{(2i+1) \cdot q\pi}{16} \right] = 1 \qquad if \ p = q$$

2D Discrete Cosine Transform (2D DCT):

$$F(u,v) = \frac{C(u)C(v)}{4} \sum_{i=0}^{7} \sum_{j=0}^{7} \cos\frac{(2i+1)u\pi}{16} \cos\frac{(2j+1)v\pi}{16} f(i,j)$$

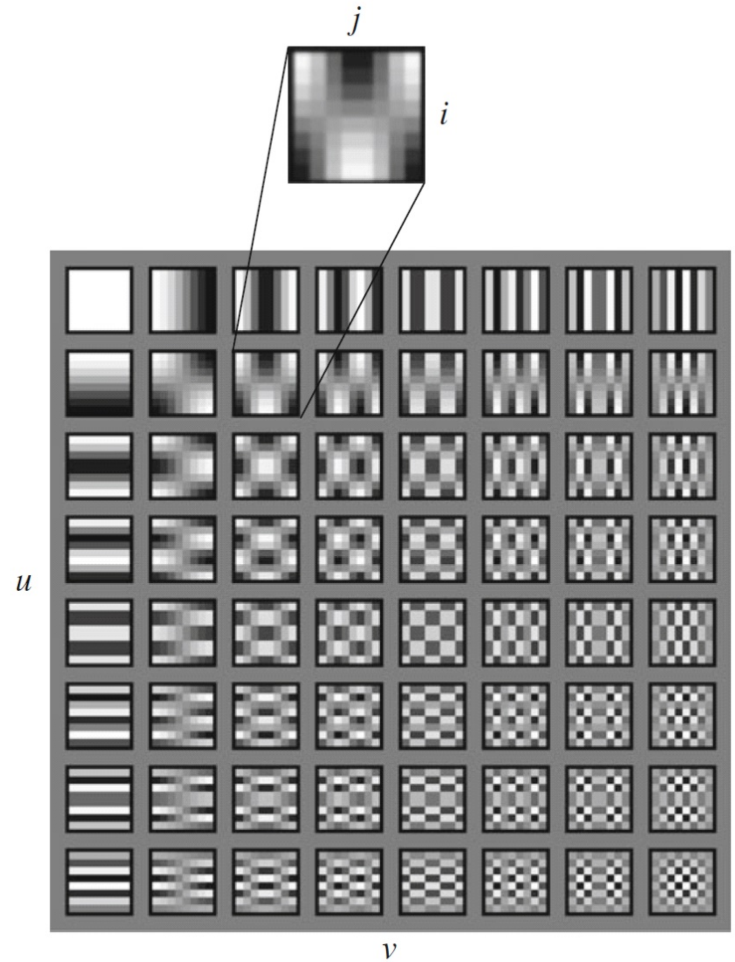
- where i, j, u, v = 0, 1, . . . , 7, and the constants $C(u)$ and $C(v)$ are determined by Eq. (8.5.16).

2D Inverse Discrete Cosine Transform (2D IDCT):

- The inverse function is almost the same, with the roles of $f(i, j)$ and $F(u, v)$ reversed, except that now $C(u)C(v)$ must stand inside the sums:

$$\tilde{f}(i,j) = \sum_{u=0}^{7} \sum_{v=0}^{7} \frac{C(u)C(v)}{4} \cos\frac{(2i+1)u\pi}{16} \cos\frac{(2j+1)v\pi}{16} F(u,v)$$

- where i, j, u, v = 0, 1, . . . , 7.



- Fig. 8.9: Graphical Illustration of 8 × 8 2D DCT basis.

2D DCT Matrix Implementation

- The above factorization of a 2D DCT into two 1D DCTs can be implemented by two consecutive matrix multiplications:

$$F(u, v) = \mathbf{T} \cdot f(i, j) \cdot \mathbf{T}^T. \tag{8.27}$$

- We will name $\mathbf{T}$ the *DCT-matrix*.

$$\mathbf{T}[i, j] = \begin{cases} \frac{1}{\sqrt{N}}, & \text{if } i = 0 \\ \sqrt{\frac{2}{N}} \cdot \cos\frac{(2j+1) \cdot i\pi}{2N}, & \text{if } i > 0 \end{cases} \tag{8.28}$$

Where *i* = 0, … , *N-1* and *j* = 0, … , *N*-1 are the row and column indices, and the block size is *N* x *N*.

- When $N = 8$, we have:

$$\mathbf{T_8}[i, j] = \begin{cases} \frac{1}{2\sqrt{2}}, & \text{if } i = 0 \\ \frac{1}{2} \cdot \cos\frac{(2j+1)\cdot i\pi}{16}, & \text{if } i > 0. \end{cases} \quad (8.29)$$

$$\mathbf{T_8} = \begin{bmatrix} \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & \frac{1}{2\sqrt{2}} & \cdots & \frac{1}{2\sqrt{2}} \\ \frac{1}{2}\cdot\cos\frac{\pi}{16} & \frac{1}{2}\cdot\cos\frac{3\pi}{16} & \frac{1}{2}\cdot\cos\frac{5\pi}{16} & \cdots & \frac{1}{2}\cdot\cos\frac{15\pi}{16} \\ \frac{1}{2}\cdot\cos\frac{\pi}{8} & \frac{1}{2}\cdot\cos\frac{3\pi}{8} & \frac{1}{2}\cdot\cos\frac{5\pi}{8} & \cdots & \frac{1}{2}\cdot\cos\frac{15\pi}{8} \\ \frac{1}{2}\cdot\cos\frac{3\pi}{16} & \frac{1}{2}\cdot\cos\frac{9\pi}{16} & \frac{1}{2}\cdot\cos\frac{15\pi}{16} & \cdots & \frac{1}{2}\cdot\cos\frac{45\pi}{16} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{1}{2}\cdot\cos\frac{7\pi}{16} & \frac{1}{2}\cdot\cos\frac{21\pi}{16} & \frac{1}{2}\cdot\cos\frac{35\pi}{16} & \cdots & \frac{1}{2}\cdot\cos\frac{105\pi}{16} \end{bmatrix}. \quad (8.30)$$

2D IDCT Matrix Implementation

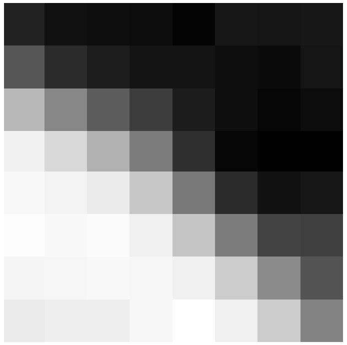
- The 2D IDCT matrix implementation is simply:

$$f(i, j) = \mathbf{T}^T \cdot F(u, v) \cdot \mathbf{T}. \qquad (8.31)$$

- See the textbook for step-by-step derivation of the above equation.
  - The key point is: the DCT-matrix is orthogonal, hence, $\mathbf{T}^T = \mathbf{T}^{-1}.$

2-D 8-point DCT Example

- Original Data:



89	78	76	75	70	82	81	82
122	95	86	80	80	76	74	81
184	153	126	106	85	76	71	75
221	205	180	146	97	71	68	67
225	222	217	194	144	95	78	82
228	225	227	220	193	146	110	108
223	224	225	224	220	197	156	120
217	219	219	224	230	220	197	151

- 2-D DCT Coefficients (after rounding to integers):

Most energy is in the upper-left corner

1155	259	-23	6	11	7	3	0
-377	-50	85	-10	10	4	7	-3
-4	-158	-24	42	-15	1	0	1
-2	3	-34	-19	9	-5	4	-1
1	9	6	-15	-10	6	-5	-1
3	13	3	6	-9	2	0	-3
8	-2	4	-1	3	-1	0	-2
2	0	-3	2	-2	0	0	-1